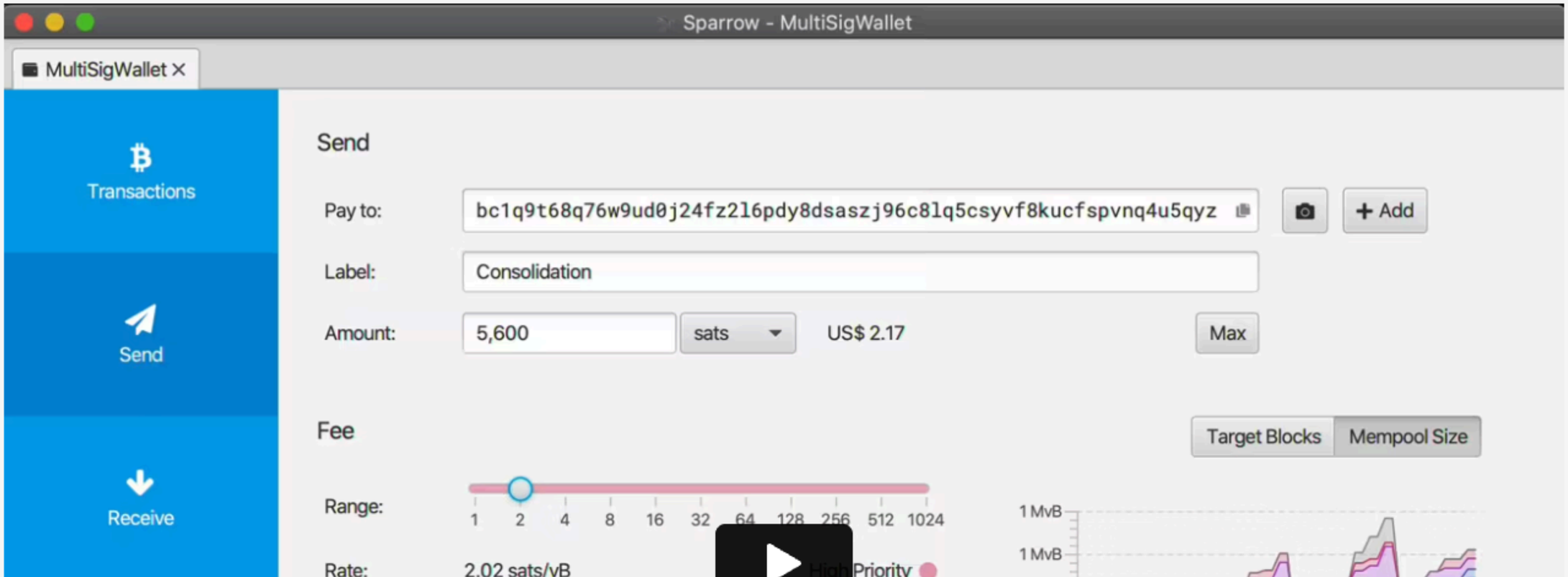
bitcoin++

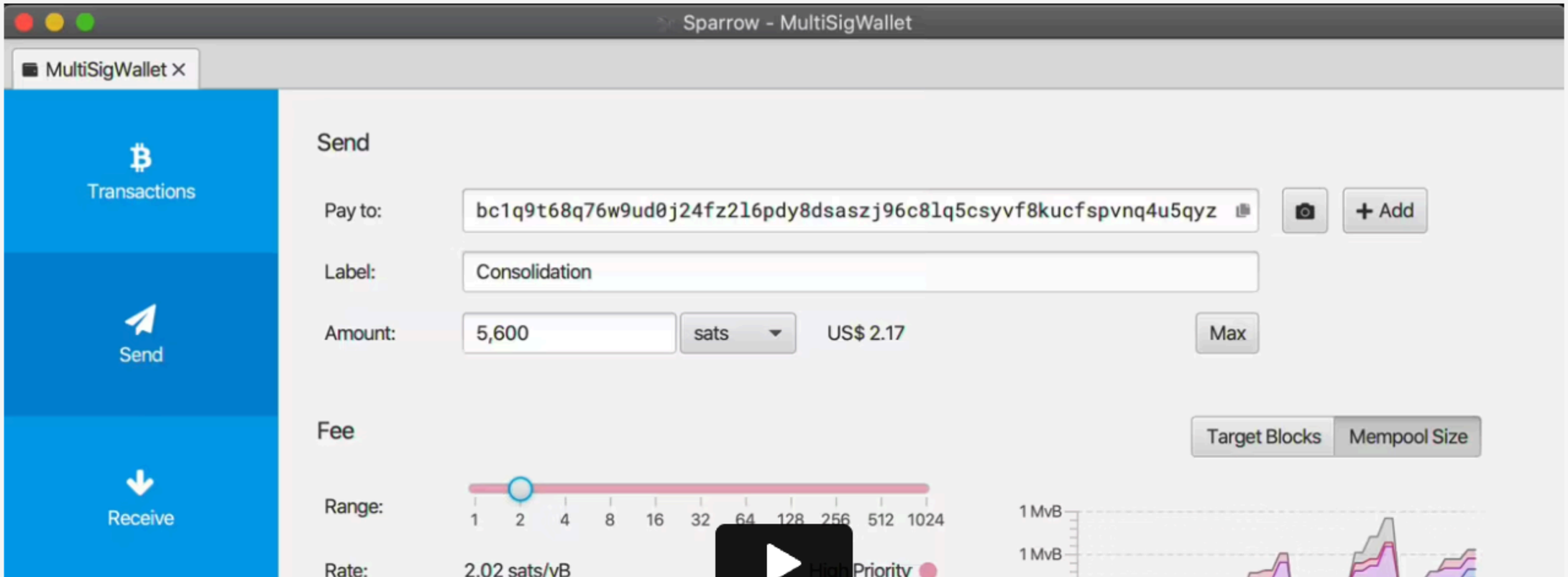
Sparrow Bitcoin Wallet

Sparrow is a Bitcoin wallet for those who value financial self sovereignty. Sparrow's emphasis is on security, privacy and usability. Sparrow does not hide information from you - on the contrary it attempts to provide as much detail as possible about your transactions and UTXOs, but in a way that is manageable and usable.



Sparrow Bitcoin Wallet

Sparrow is a Bitcoin wallet for those who value financial self sovereignty. Sparrow's emphasis is on security, privacy and usability. Sparrow does not hide information from you - on the contrary it attempts to provide as much detail as possible about your transactions and UTXOs, but in a way that is manageable and usable.



Silent payments output descriptor BIP proposal

Inbox ×

Craig Raw <craigraw@gmail.com>
to me

Dec 12, 2025, 5:33 PM (4 days ago)

Hey Lisa,

Great hanging out last weekend!

As promised here's a quick summary of my silent payments output descriptor BIP, mostly lifted from my mail to bitcoin-dev:

There is a practical need for a silent payments output descriptor format in order to enable wallet interoperability and backup/recovery. There has been sc
discussion on this topic [1][2] which this BIP proposal builds on:

https://github.com/bitcoin/bips/pull/2047

In summary a new top level script expression sp() is defined, which takes as it's first argument one of two new key expressions:

- spscan1q... which encodes the scan private key and the spend public key
- spspend1q... which encodes the scan private key and the spend private key

The outputs may then be generated by combining this key material with the sender input public keys. As suggested by the naming, spscan key expressi
scanning only (watch only) wallets, while spspend key expressions allow you to reconstruct a wallet that can spend.

In order to reduce the scanning burden, a block height may be optionally specified in the sp() expression as a second argument for a wallet birthday. Fin
more positive integers may be specified as further arguments to scan for additional BIP352 labels. The change label (m = 0) is implicitly included.

nts output descriptor BIP proposal Inbox ×

@gmail.com> Dec 12, 2025, 5:33 PM (4 days ago)

weekend!

quick summary of my silent payments output descriptor BIP, mostly lifted from my mail to bitcoin-dev:

ed for a silent payments output descriptor format in order to enable wallet interoperability and backup/recovery. There has been sc
c [1][2] which this BIP proposal builds on:

oin/bips/pull/2047

level script expression sp() is defined, which takes as it's first argument one of two new key expressions:

which encodes the scan private key and the spend public key
which encodes the scan private key and the spend private key

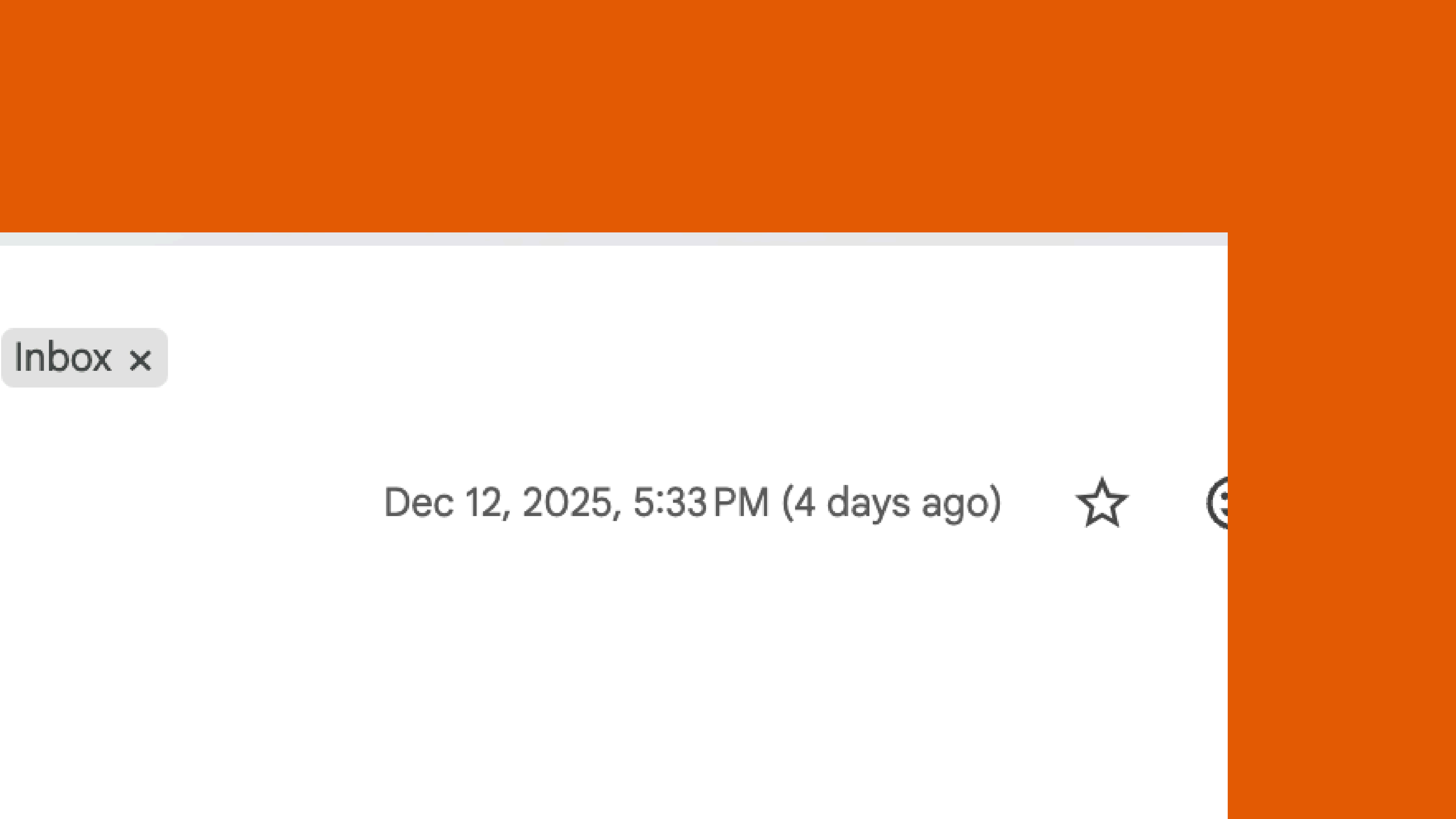
be generated by combining this key material with the sender input public keys. As suggested by the naming, spscan key expressi

or BIP proposal Inbox ×

Dec 12, 2025, 5:33 PM (4 days ago)

ents output descriptor BIP, mostly lifted from my mail to bitcoin-dev:

escriptor format in order to enable wallet interoperability and backup/recovery. There has been sc
ilds on:


Inbox ×
Dec 12, 2025, 5:33 PM (4 days ago)

2025, 5:33 PM (4 days ago)

2025, 5:33 PM (4 days ago)

There is a practical need for a silent payments output descriptor format in order to enable wallet interoperability and backup/recovery. There has been some prior discussion on this topic [1][2] which this BIP proposal builds on:

There is a practical need for a silent payments output descriptor format in order to enable wallet interoperability and backup/recovery. There has been some prior discussion on this topic [1][2] which this BIP proposal builds on:

There is a practical need for a silent payments output descriptor format in order to enable wallet interoperability and backup/recovery. There has been some prior discussion on this topic [1][2] which this BIP proposal builds on:

There is a practical need for a silent payments output descriptor format in order to enable wallet interoperability and backup/recovery. There has been some prior discussion on this topic [1][2] which this BIP proposal builds on:

bitcoin / bips

Code | Pull requests 67 | Actions | Wiki | Security | Insights

Add sp() output descriptor format for BIP352 Silent Payments

Open craigraw wants to merge 2 commits into `bitcoin:master` from `craigraw:spdescriptor`

Conversation 7 | Commits 2 | Checks 4 | Files changed 1

Commit 6807d2f

Filter files...

bip-spdescriptor.mediawiki

Update headers and remove space after comma in descriptors

craigraw committed 2 weeks ago · ✓ 4 / 4

bip-spdescriptor.mediawiki

Specification

A new top level script expression is defined: `sp()`.

Key Expressions

Two new key expression types are defined for use with `sp()` descriptors:

`spscan`

The `spscan` key expression encodes the scan private key and spend public

- The human-readable part "spscan" for mainnet, "tspscan" for testnets
- The data-part values:

Specification

A new top level script expression is defined: `sp()`.

Key Expressions

Two new key expression types are defined for use with `sp()` descriptors:

`spscan`

The `spscan` key expression encodes the scan private key and spend public

- The human-readable part "spscan" for mainnet, "tspscan" for testnets
- The data-part values:

Specification

A new top level script expression is defined: `sp()`.

Key Expressions

Two new key expression types are defined for use with `sp()` descriptors:

`spscan`

The `spscan` key expression encodes the scan private key and spend public

- The human-readable part "spscan" for mainnet, "tspscan" for testnets
- The data-part values:

Key Expressions

Two new key expression types are defined for

`spscan`

The `spscan` key expression encodes the scan

- The human-readable part "spscan" for m
- The data-part values:
  - The character "q", to represent silent
  - The payload: `ser<sub>256</sub>(b<`

`spspend`

The `spspend` key expression encodes both th

Key Expressions

Two new key expression types are defined for

`spscan`

The `spscan` key expression encodes the scan

- The human-readable part "spscan" for m
- The data-part values:
  - The character "q", to represent silent
  - The payload: `ser<sub>256</sub>(b<`

`spspend`

The `spspend` key expression encodes both th

`spscan`

The `spscan` key expression encodes the scan

- The human-readable part "spscan" for m
- The data-part values:
  - The character "q", to represent silent
  - The payload: `ser<sub>256</sub>(b<`

`spspend`

The `spspend` key expression encodes both th

- The human-readable part "spspend" for
- The data-part values:
  - The character "q", to represent silent

`spscan`

The `spscan` key expression encodes the scan

- The human-readable part "spscan" for m
- The data-part values:
  - The character "q", to represent silent
  - The payload: `ser<sub>256</sub>(b<`

`spspend`

The `spspend` key expression encodes both th

- The human-readable part "spspend" for
- The data-part values:
  - The character "q", to represent silent

`spscan`

The `spscan` key expression encodes the scan private key and spend public ke

- The human-readable part "spscan" for mainnet, "tspscan" for testnets
- The data part values:

`spscan`

The `spscan` key expression encodes the scan private key and spend public ke

- The human-readable part "spscan" for mainnet, "tspscan" for testnets
- The data part values:

`spscan`

The `spscan` key expression encodes the scan private key and spend public ke

- The human-readable part "spscan" for mainnet, "tspscan" for testnets
- The data part values:

`spspend`

The `spspend` key expression encodes both the scan and spend private keys. It is

- The human-readable part "spspend" for mainnet, "tspspend" for testnets

`spspend`

The `spspend` key expression encodes both the scan and spend private keys. It is

- The human-readable part "spspend" for mainnet, "tspspend" for testnets

`spspend`

The `spspend` key expression encodes both the scan and spend private keys. It is

- The human-readable part "spspend" for mainnet, "tspspend" for testnets

the descriptor must be scanned for when detecting payments.

For watch-only wallets, use `spscan` encoding. For full wallets that can both scan and spend, use `spspend` encodi

Backwards Compatibility

the descriptor must be scanned for when detecting payments.

For watch-only wallets, use `spscan` encoding. For full wallets that can both scan and spend, use `spspend` encodi

Backwards Compatibility

`spscan`

The `spscan` key expression encodes the scan private key and spend public

- The human-readable part "spscan" for mainnet, "tspscan" for testnets

the descriptor must be scanned for when detecting payments.

For watch-only wallets, use `spscan` encoding. For full wallets that can both scan and spend, use `spspend` encodi

Backwards Compatibility

`spscan`

The `spscan` key expression encodes the scan private key and spend public

- The human-readable part "spscan" for mainnet, "tspscan" for testnets

the descriptor must be scanned for when detecting payments.

For watch-only wallets, use `spscan` encoding. For full wallets that can both scan and spend, use `spspend` encodi

Backwards Compatibility

`spscan`

The `spscan` key expression encodes the scan private key and spend public

- The human-readable part "spscan" for mainnet, "tspscan" for testnets

the descriptor must be scanned for when detecting payments.

For watch-only wallets, use `spscan` encoding. For full wallets that can both scan and spend, use `spspend` encodi

Backwards Compatibility

the descriptor must be scanned for when detecting payments.

For watch-only wallets, use `spscan` encoding. For full wallets that can both scan and spend, use `spspend` encodi

Backwards Compatibility

the descriptor must be scanned for when detecting payments.

For watch-only wallets, use `spscan` encoding. For full wallets that can both scan and spend, use `spspend` encodi

Backwards Compatibility

`spspend`

The `spspend` key expression encodes both the scan and spend private keys. It is

- The human-readable part "spspend" for mainnet, "tspspend" for testnets

the descriptor must be scanned for when detecting payments.

For watch-only wallets, use `spscan` encoding. For full wallets that can both scan and spend, use `spspend` encodi

Backwards Compatibility

`spspend`

The `spspend` key expression encodes both the scan and spend private keys. It is

- The human-readable part "spspend" for mainnet, "tspspend" for testnets

the descriptor must be scanned for when detecting payments.

For watch-only wallets, use `spscan` encoding. For full wallets that can both scan and spend, use `spspend` encodi

Backwards Compatibility

`spspend`

The `spspend` key expression encodes both the scan and spend private keys. It is

- The human-readable part "spspend" for mainnet, "tspspend" for testnets

Specification

A new top level script expression is defined: `sp()`.

Key Expressions

Two new key expression types are defined for use with `sp()` descriptors:

`spscan`

The `spscan` key expression encodes the scan private key and spend public

- The human-readable part "spscan" for mainnet, "tspscan" for testnets
- The data-part values:

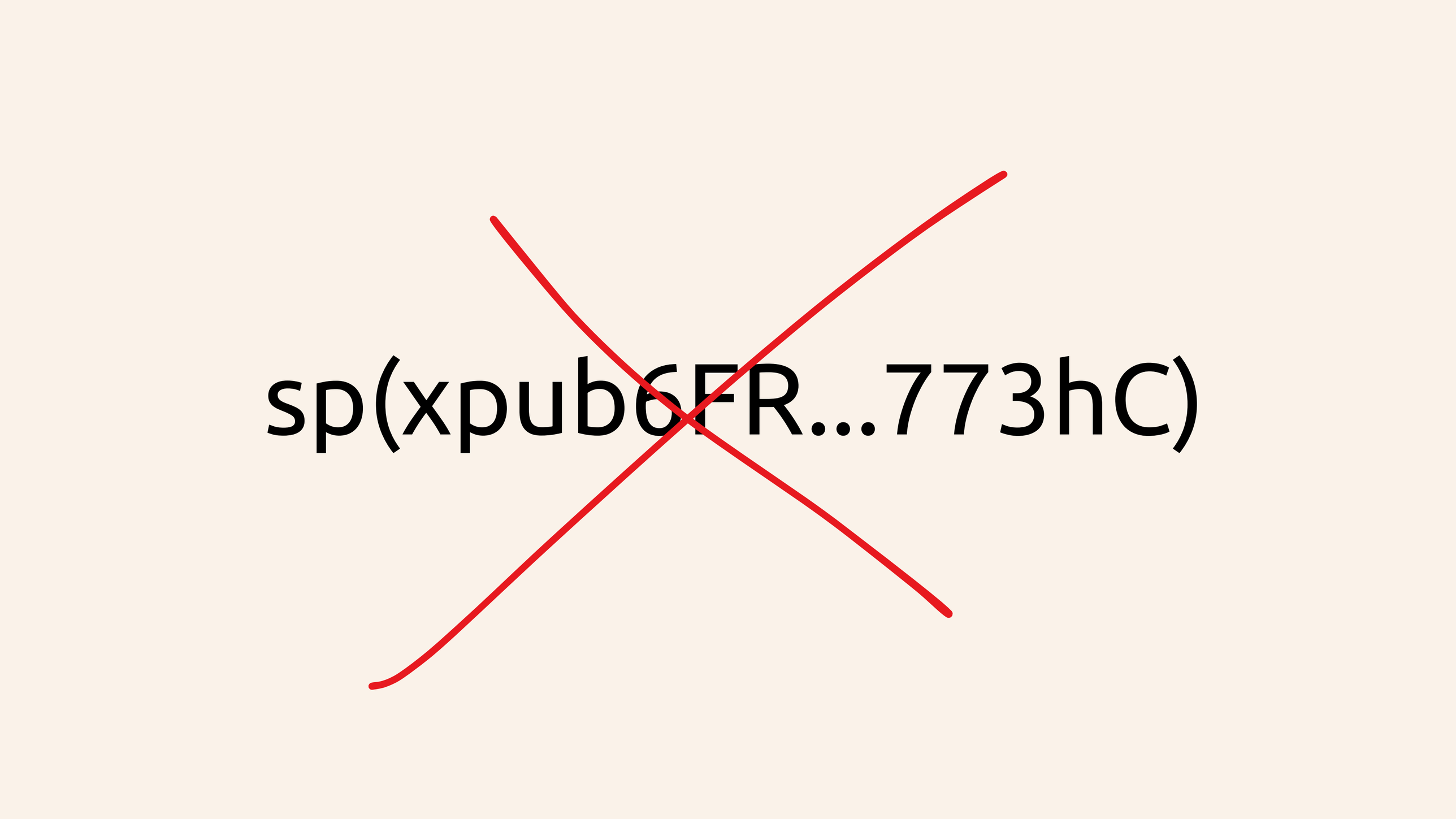
Specification

A new top level script expression is defined: `sp()`.

sp()

sp([key expression])

sp(xpub6FR...773hC)


sp(xpub6FR...773hC)

sp([key expression])

silent payment

sp([key expression])

sp(spscan1q...)

sp(spspend1q...)

sp(spspend1q...)

`spscan`

The `spscan` key expression encodes the scan

- The human-readable part "spscan" for m
- The data-part values:
  - The character "q", to represent silent
  - The payload: `ser<sub>256</sub>(b<`

`spspend`

The `spspend` key expression encodes both th

- The human-readable part "spspend" for
- The data-part values:
  - The character "q", to represent silent

`spscan`

The `spscan` key expression encodes the scan

- The human-readable part "spscan" for m
- The data-part values:
  - The character "q", to represent silent
  - The payload: `ser<sub>256</sub>(b<`

`spspend`

The `spspend` key expression encodes both th

- The human-readable part "spspend" for
- The data-part values:
  - The character "q", to represent silent

`spscan`

The `spscan` key expression encodes the scan

- The human-readable part "spscan" for m
- The data-part values:
  - The character "q", to represent silent
  - The payload: `ser<sub>256</sub>(b<`

`spspend`

The `spspend` key expression encodes both th

- The human-readable part "spspend" for
- The data-part values:
  - The character "q", to represent silent

sp(spscan1q...)

sp(spscan1q<private key>...)

scan key

sp(spscan1q<private key>...)

scan key

sp(spscan1q<private key>...)

32 bytes

sp(spscan1q<private key>
<public key>)

spend key

sp(spscan1q<private key>
<public key>)

spend key

sp(spscan1q<private key>
<public key>)

spend key

sp(spscan1q<private key>
<public key>)

33-byte

spend key

sp(spscan1q<private key>
<public key>)

x, y' 33-byte

sp(spscan1q<private key>
<public key>)

sp(spscan1q<private key>
<public key>)

sp(spspend1q<private key><public key>)

sp(spspend1q<private key><public key>)

sp(spspend1q<private key><private key>)

sp(spspend1q<private key><private key>)

sp(spspend1q<private key><private key>)

sp(spspend1q...)

sp(spspend1q...)

i can spend your money

sp(spscan1q...)

i can see your money

sp([key expression])

sp([key expression],
BIRTHDAY,LABEL(s))

sp([key expression],

BIRTHDAY,LABEL(s))

sp([key expression],
<block height>,LABEL(s))

```
sp([key expression],
    840000,LABEL(s))
```

```
sp([key expression],
    840000,LABEL(s))
```

```
sp([key expression],
    840000,LABEL(s))
```

sp(spscan1q...,840000)

sp([key expression],
BIRTHDAY,LABEL(s))

sp([key expression],
BIRTHDAY,LABEL(s))

```
BIP: 352
Layer: Applications
Title: Silent Payments
Author: josibake <josibake@protonmail.com>
        Ruben Somsen <rsomsen@gmail.com>
Comments-URI: https://github.com/bitcoin/bips/wiki/Comments:BIP-
Status: Proposed
Type: Standards Track
Created: 2023-03-09
License: BSD-2-Clause
```

Labels

For a single silent payment address of the form (B_{scan}, B_{spend}), Bob may wish to differentiate in multiple silent payment addresses, but this would require him to scan for each one, which becc can label his spend public key B_{spend} with an integer m in the following way:

- Let $B_m = B_{spend} + hash(b_{scan} || m) \cdot G$ where m is an incrementable integer starting from 1
- Publish (B_{scan}, B_1), (B_{scan}, B_2) etc.

Alice performs the tweak as before using one of the published (B_{scan}, B_m) pairs. Bob detects th

Labels

For a single silent payment address of the form (B_{scan}, B_{spend}), Bob may wish to differentiate in multiple silent payment addresses, but this would require him to scan for each one, which beco can label his spend public key B_{spend} with an integer m in the following way:

- Let $B_m = B_{spend} + hash(b_{scan} || m) \cdot G$ where m is an incrementable integer starting from 1
- Publish (B_{scan}, B_1), (B_{scan}, B_2) etc.

Alice performs the tweak as before using one of the published (B_{scan}, B_m) pairs. Bob detects th

Labels

For a single silent payment address of the form (B_{scan}, B_{spend}), Bob may wish to differentiate in multiple silent payment addresses, but this would require him to scan for each one, which becc can label his spend public key B_{spend} with an integer m in the following way:

- Let $B_m = B_{spend} + hash(b_{scan} || m) \cdot G$ where m is an incrementable integer starting from 1
- Publish (B_{scan}, B_1), (B_{scan}, B_2) etc.

Alice performs the tweak as before using one of the published (B_{scan}, B_m) pairs. Bob detects th

Labels

For a single silent payment address of the form (B_{scan}, B_{spend}), Bob may wish to differentiate in
multiple silent payment addresses, but this would require him to scan for each one, which beco
can label his spend public key B_{spend} with an integer m in the following way:

- Let $B_m = B_{spend} + hash(b_{scan} || m) \cdot G$ where m is an incrementable integer starting from 1
- Publish (B_{scan}, B_1), (B_{scan}, B_2) etc.

Alice performs the tweak as before using one of the published (B_{scan}, B_m) pairs. Bob detects th

B_{spend}), Bob may wish to differentiate incoming payments. Naively,
ire him to scan for each one, which becomes prohibitively expensiv
in the following way:

incrementable integer starting from 1

It is important to note that an outside observer can easily deduce th
published address will have B_{scan} in common. As such, labels are no
way for Bob to determine the source of an incoming payment.

Labels for change

It is important to note that an outside observer can easily deduce th
published address will have B_{scan} in common. As such, labels are no
way for Bob to determine the source of an incoming payment.

Labels for change

sp([key expression],
BIRTHDAY,LABEL(s))

sp([key expression],
BIRTHDAY,”janet”)

sp([key expression],
BIRTHDAY,”janet”, “nifty”)

Labels

For a single silent payment address of the form (B_{scan}, B_{spend}), Bob may wish to differentia
multiple silent payment addresses, but this would require him to scan for each one, which
can label his spend public key B_{spend} with an integer m in the following way:

- Let $B_m = B_{spend} + hash(b_{scan} || m) \cdot G$ where m is an incrementable integer starting fro
- Publish (B_{scan}, B_1), (B_{scan}, B_2) etc.

Labels

For a single silent payment address of the form (B_{scan}, B_{spend}), Bob may wish to differentia
multiple silent payment addresses, but this would require him to scan for each one, which
can label his spend public key B_{spend} with an integer m in the following way:

- Let $B_m = B_{spend} + hash(b_{scan} || m) \cdot G$ where m is an incrementable integer starting fro
- Publish (B_{scan}, B_1), (B_{scan}, B_2) etc.

Labels

For a single silent payment address of the form (B_{scan}, B_{spend}), Bob may wish to differentia
multiple silent payment addresses, but this would require him to scan for each one, which
can label his spend public key B_{spend} with an integer m in the following way:

- Let $B_m = B_{spend} + hash(b_{scan} || m) \cdot G$ where m is an incrementable integer starting fro
- Publish (B_{scan}, B_1), (B_{scan}, B_2) etc.

sp([key expression],
BIRTHDAY,LABEL(s))

sp([key expression],
BIRTHDAY,1)

sp([key expression],
BIRTHDAY,1,10)

sp([key expression],
BIRTHDAY,1,10,21)

sp(spscan1q...,840000)

sp(spscan1q...,840000)

sp(spscan1q...,840000,1,10,21

sp(spscan1q...,840000,1,10,21)

akes:

- A key expression (first argument)
- A birthday as a positive integer representing a block height (second argument)
- Zero or more label integers (remaining arguments), where each label is a positive integer

Code | Pull requests 67 | Actions | Wiki | Security | Insights

Add sp() output descriptor format for BIP352 Silent Payments #2047

Open craigraw wants to merge 2 commits into `bitcoin:master` from `craigraw:spdescriptor`

Conversation 7 | Commits 2 | Checks 4 | Files changed 1

Commit 6807d2f | 0 / 1 viewed

Filter files...

bip-spdescriptor.mediawiki

Update headers and remove space after comma in descriptors

craigraw committed 2 weeks ago · ✓ 4 / 4

bip-spdescriptor.mediawiki

```
BIP: 352
Layer: Applications
Title: Silent Payments
Author: josibake <josibake@protonmail.com>
        Ruben Somsen <rsomsen@gmail.com>
Comments-URI: https://github.com/bitcoin/bips/wiki/Comments:BIP-
Status: Proposed
Type: Standards Track
Created: 2023-03-09
License: BSD-2-Clause
```

Edit profile

nifty, btc++ taipei dec 15-17

@niftynei

irreverent orange-pilled postrat. teaches @base58btc writes @btcinsider__ hosts @btcplusplus runs @core_LN ⚡ eng @blockstream

ATX btcpp.dev Joined March 2009 >

Edit profile

nifty. btc++ taipei dec 15-17

@niftynei

irreverent orange-pilled postrat. teaches @base58btc writes @btcinsider__
hosts @btcplusplus runs @core_LN ⚡ eng @blockstream

ATX btcpp.dev Joined March 2009 >

Base58

School of Engineering

a protocol school built on the bitcoin standard

base58.school

bitcoin++

主權版